Extracting predicate structures from parse trees

Klenner, M

Extracting Predicate Structures from Parse Trees

Manfred Klenner
Institute of Computational Linguistics
University of Zurich
klenner@cl.unizh.ch

Abstract

We evaluate two different approaches for extracting predicate structures from parse trees. We compare the results of a rule-based algorithm incorporating decision tree learning to an integer linear programming (ILP) approach with an underlying statistical model. It turns out that the rule-based approach yields higher precision but lower recall than the ILP approach. Both approaches achieve precision rates of more than 90 %.

1 Introduction

Recently, much attention has been paid to semantic role labeling (e.g. the CoNLL-2004 and CoNLL-2005 shared task). The task is to identify the semantic roles of a verb, i.e. which phrase (e.g. NP, PP, S) realizes which semantic role (e.g. agent, patient). A related problem is grammatical relation finding (e.g. identifying the subject of a verb) which can be done as part of parsing or as a separate process on top of parse trees or chunks (Buchholz, 2002). While for some applications semantic role labeling goes too far, grammatical role finding is not sufficient. Especially in logic based approaches, predicate structures are more common than frames with semantic roles. An example of a predicate structure is: believe(peter,like(mary,books_of(max_frisch))). Of course, these structures can, in principle, be derived from semantic role labeled verb frames, but the question is whether there is a more direct way. Grammatical relations (GR), on the other hand, provide useful information to support predicate argument extraction, if a mapping from grammatical relations to argument positions of verb predicates is assumed. For example: the subject of an active verb is mapped to the first argument position of the underlying verb predicate.

Syntactic parsing and grammatical relation finding for unrestricted text requires robust, statistical approaches. The resulting structures (e.g parse trees) are noisy: tagging errors, attachment mistakes and wrong case assignments (i.e. grammatical relation identification is false) are to be expected. As a consequence, a robust method for the extraction of predicate structures from parse trees is needed as well.

2 Tools and Resources

We use the BitPar parser (Schmid , 2004) and a treebank grammar (Schiehlen , 2004) derived from the Negra corpus (Brants et al., 1999), a German tree bank of 20.000 sentences. Although the grammar does not specify GR, case is assigned to noun phrases. The case feature serves as an indicator of GR (e.g. nominative case indicates subject). Note that clauses (e.g. complement clauses) do not bear case, the decision whether they are verb complements (and thus arguments of predicates) or not, cannot be drawn from functional information, thus.

In our experiments, we used 1001 manually extracted predicate structures (the gold standard) derived from 870 sentences. There were 16 three-placed predicates, 512 two-placed and 454 one-placed. Because of the low frequency, we omitted the three-placed predicates from our experiments. That is, no rules are being learned for three-placed predicates.

Since the parser does not identify heads, we defined a head heuristic. Its precision is $> 99\%$. Given the 2543 heads in our corpus of 870 sentences, 25 head assignments are wrong (in the worst case). Note, that these mistakes propagate to the precision of the rule learner and the ILP approach.

3 The Problem of Argument Assignment

There are two problems to be solved: an identification problem (which heads of which phrases

are arguments) and an assignment problem (which argument position do they fill). Input is a parse tree, output is a predicate structure. For example, the sentence "Das Volk laesst die Scharia kalt (Sharia law leaves people cold)" is mapped onto "kalt_lassen(scharia,volk)" (leaves_cold(sharia, people)). As previously mentioned, the statistical parse is imperfect: sometimes there are more e.g. nominative heads than possible subjects. Sometimes case assignment is wrong (e.g accusative instead of nominative) or case is even missing. Especially, verbal heads never get case but they are potential verb arguments (complement clauses). There are tagging errors (e.g. a head is tagged as a non-head category) and also attachment mistakes (dislocated complements) are taking place.

In other words, there is noise in the data, and a statistical or machine learning approach could help to recover from those flaws.

4 Learning Interpretation Rules

Information in parse trees is structurally encoded. To extract it, the structural patterns need to be identified. This could be done either manually (writing semantic interpretation rules) or automatically. In both cases, reliable data (a gold standard) for evaluation purposes is needed. The second variant, however, has some obvious advantages. Extraction rules must be tailored to the tree format produced by the parser. If in the course of the lifetime of a system the parser is replaced by a better one (or a better version of the old) new interpretation rules must be either manually written or automatically derived. The later alternative is clearly preferable. But there is another reason why to prefer the second solution: the noise. Machine Learning approaches are better than humans to cope with noisy data (at least given mass data).

Given a set of syntax trees produced by a statistical parser and given a gold standard of manually extracted predicate structures that corresponds to these parse trees, interpretation rules can be learned by a simple procedure. Each predicate structure unambiguously identifies a verb (via the predicate name) and the complements of the verb (via the predicate arguments). The basic rule learning algorithm is as follows:

starting from the verb node in the syntax tree

- search for an *anchor* node, i.e., a predecessor of the verb node (often its mother) which dominates all verb complements
- save the paths from the anchor to the complement heads in a left to right order
- save the features of each node of the path (e.g. syntactic label, case)
- save the mapping (how is the linear order of the parse tree projected onto the order of predicate arguments)

Assume the (partial) syntax tree given in Fig. 1 and its gold standard predicate structure " bearbeite(er,Konzerte)" (adapt(he, concerts)).

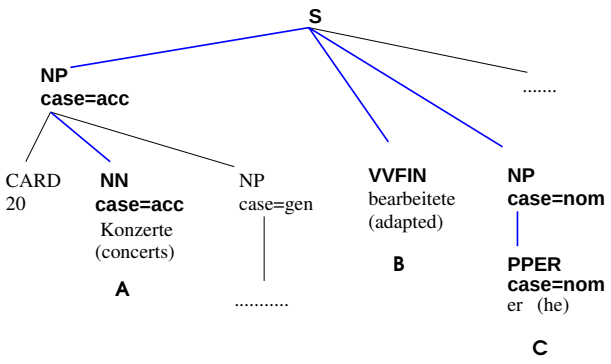

Figure 1: Fragment Indicating a Rule Pattern

The direct object ($concerts$) precedes the verb, the subject (he) follows the verb. 'S' is the anchor node and there are three paths connecting the anchor to the heads (including the verb). The underlying structural pattern, the extraction rule derived from that positive example, is highlighted (bold) - see Fig. 2 for a rule representation. The anchor node has category S. It is the root of three paths: A,B,C. NN, VVFIN and PPER

```
anchor: S
A=[NP, NN with case=acc]
B=[VVFIN]
C=[NP, PPER with case=nom]
Linear Precedence: A < B < C
Mapping: B(C,A)
```

Figure 2: Rule Representation

```
A=[NP, {PPER|NN|PRF} with case=acc]
B=[VVFIN]
C=[NP,{PPER|NN} with case=nom]
```

Figure 3: Generalized Rule

are the leave nodes. The words attached to these nodes form the arguments of the predicate. *Linear Precedence* fixes the order in which these paths are given in the parse tree and *Mapping* is used to construct the predicate structure from the words at the leave nodes.

5 Evaluation

We run the algorithm on the 870 sentences Input was the set of parse trees, output were the learned rules. 223 rules were generated in the training set at each run (on the average). There were 147 (idiosyncratic) rules stemming from exactly 1 positive examples. The rest of the rules covers 2 or more examples (up to 70 per rule). On the average, every rule thus covers four positive examples. That is a poor verbs per rule ratio. We found however that these rules often are just minor variants of each other. We implemented a rule generalization component that reduced the 223 rules to 81 (leaving 29 idiosyncratic rules). The generalized rules assemble structural patterns with identical paths, but different categorical realizations of leave nodes. Fig. 3 shows a generalized rule covering all the categorical variants of the rule from Fig. 2. E.g., the leave of path A might be a personal pronoun (PPER), a normal noun (NN) or a reflexive pronoun (PRF).

We evaluated precision and recall on the training set and test set for 1-ary and 2-ary predicates, respectively (see Fig. 4). Input was the set of parse trees together with the learned rules, output were the predicate argument structures found.

	Prec_{train}	Rec_{train}	Prec_{test}	Rec_{test}
2	84.2	98.4	79.1	70.9
1	99.8	99.7	99.3	86.6

Figure 4: Evaluation Results

First of all, precision on the 2-ary predicates in the training set is low (84.2 %). There are various reasons for this. As already mentioned, errors stemming from the head heuristic propagate to the rule learner. If the wrong head (of a np) is chosen, the predicate structure will have an incorrect argument. Moreover, parsing errors might result in erroneous extraction rules. To give an example consider the wrong (case) parse: "Das $\text{Volk}_{[nom]}$ laesst die $\text{Scharia}_{[acc]}$ kalt (Sharia law leaves people cold). "Das Volk (people)" is the direct object (accusative case). However, the parser attached the nominative case. Actually, this is a morphologically licensed assignment. "Das Volk" and "die Scharia" can be nominative or accusative, respectively. Since it is more likely to have the subject (nominative) preceding the verb, the parser did a reasonable but erroneous job by assigning nominative case to "Das Volk". Since the gold standard predicate structure, i.e. leave_cold(sharia,people), has *people* as the second argument, a rule is generated that maps the head of a nominative np to the second argument position of the verb predicate. This way contradicting rules are generated: one that maps nominative to the first argument position (correct decision) and one that maps nominative to the second. Both rules have the some triggering conditions (i.e. the same paths), they produce conflicting interpretations (and reduce precision). In the experimental setting reported in Fig. 4 and Fig. 5 every rule that matches is applied. We also implemented a version of the rule learner that does rule weighting and deletes contradicting rules keeping the rules with the higher weight (see section 7).

	Prec	Rec	F-meas
train	92.0	99.1	95.4
test	89.2	78.8	83.7

Figure 5: Summary of the Results

Fig. 5 gives a summary of the results and provides the values of the f-measure. Note that a recall < 100 on the training set stems from errors of the head heuristic. Rules are learned according to the gold standard, that is, with perfect head information. But in the evaluation the head heuristic is used. If it fails, the wrong argument is extracted and precision drops down. We also defined a simple procedure to fix a base line: every verb gets as its arguments the nominative, accusative and da-

tive heads (in that order) under the anchor node (the dominating S node). If an embedded S node is present, then its head verb fills the last argument position of the predicate. The results are given in Fig. 6.

	Prec	Rec	F-meas
train	71.1	75.3	73.14
test	75.2	72.1	73.61

Figure 6: Base Line

6 Rule Specialization with Decision Trees

Precision drops, if rules classify negative instances as positive (e.g. the case of contradicting rules from the previous discussion). One way to improve rule precision is to make rules more specific. This can be accomplished with a decision tree learner. We incorporated this along the following lines: If the paths of a rule match a syntax tree, the rule is applicable. To make it more specific, a decision tree is attached to each rule to further restrict its application. The decision tree learner is trained with vectors derived from positive and negative examples of the syntax trees accepted by the rule.

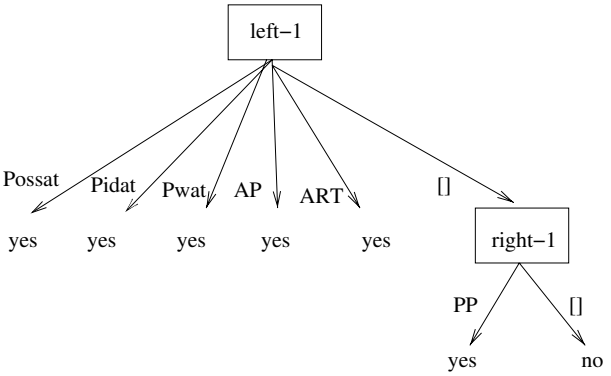

Figure 7: Decision Tree for Rule Application

Fig. 7 shows the very simple decision tree learned for the rule derived from the tree in Fig. 1. We used contextual features to specify the training vectors: the feature values (syntactic label and case) of the sister nodes (left and right neighbors) of the leave nodes of each path. In this learned decision rule, only the left and right sisters of the leave node of path A (cf. Fig. 2) come into play: *left-1* and *right-1*: if the left sister of the leave node is $Possat$ (an attributing possessive pronoun) then the rule triggers. The same is true with other pronouns ($Pidat$, $Pwat$), an adjective phrase (AP) and a determiner (ART). Only if the left (*left-1*) and the right (*right-1*) context are empty ([]), then the rule is not allowed to trigger. Such rules are not very instructive, linguistically. But they work very well (see Fig. 8). Precision goes up to >98 %, however recall drops (70.7 %). Recall drops since rules are getting more specific.

	Prec	Rec	F-meas
train	99.6	98.7	99.1
test	98.4	70.7	82.3

Figure 8: Results of the Decision Tree Version

7 Rule Weighting

The best results were achieved with a version of the rule learner based on a simple form of statistics, namely rule weighting. The measure is:

$$\frac{|\ positive\ examples\ |}{|\ positive\ and\ negative\ examples\ |}$$

If more than one rule applies to a parse tree, then only the rule with the highest score is applied. See Fig. 9 for the results.

	Prec	Rec	F-meas
train	98.9	98.6	98.7
test	98.5	78.4	87.3

Figure 9: Results of the Weighted Version

Precision and recall in the training set are excellent - the f-measure is 98.7%. Also the precision on the data of the test set is good. However, recall is still too low (78.4 %). A low recall means that the rules generated from the training set do not capture enough of the syntactic patterns needed to process the data in the test set. In other words, there is too much variance in the syntax trees - rules are missing.

One have to bear in mind that predicate extraction is simpler than semantic role labeling. Nethertheless, we have argued that a machine

learning approach is sensible, because the structures provided by current parsers for unrestricted text are noisy. The idea is to let the rule extractor learn how to cope with that noise and in the best case it should be able to correct systematic mistakes made by the parser. Our rule learner is straightforward. However, we found it interesting to compare it to the results of a more general machine learning approach. We started with TIMBL (Daelemans et al., 2004), but found it more convenient to use integer linear programming, because linguistic constraints (e.g. that a verb has, say, at most 3 arguments) can be expressed more naturally with ILP than with memory-based learners like TIMBL (where such global constraints are to be modelled as class decisions, which, at least in our experiments, results in a poor performance).

8 ILP for NLP

Integer Linear Programming (ILP) is the name of a class of constraint satisfaction algorithms which are restricted to a numerical representation of the problem to be solved. The objective is to optimize the numerical solution (the *objective function* below). Optimization means maximization or minimization of linear equations. The general form of an ILP specification is given in Fig. 10.

Objective Function:

$$maxf(X_1, \ldots, X_n) := c_1X_1 + c_2X_2 + \ldots + c_nX_n$$

Constraints:

$$a_{i1}X_1 + a_{i2}X_2 + \ldots + a_{in}X_n \left(\begin{array}{c} \leq \\ = \\ \geq \end{array} \right) b_i,$$

$i = 1, \ldots, m$

X_i are variables, c_i, b_i and a_{ij} are constants.

Figure 10: ILP Specification

The goal is to maximize a n-ary function f, which is defined (':=') as the sum of all c_iX_i. Argument assignment decisions can be modeled in the following way: X_i are binary variables that indicate the (non-)assignment of a head to an argument position of a verb. If the value of X_i is 1, the attachment was successful, otherwise ($X_i = 0$) it failed. c_i and a_{ij} are weights that represent the impact of an assigment; they provide an empirically based numerical justification of the assignment. Finally, the variables b_i are used to restrict the number of X_i that are to be chosen (in our model, a verb predicate can have at most 3 argument positions).

To our knowledge, (Punyakanok et al., 2004) were the first who applied ILP to NLP. Their treatment of semantic role labeling shares some similarities with our approach, however there are differences (see related work).

Given a sentence with a number of verbs v ($v \geq 1$) and a number of heads h (nominal categories or verbs) where $h \geq v$.

1. Determine for each verb the number of arguments it has.

2. Choose for each argument position of a (verb) predicate a head that fills it.

Since $v \geq 1$, variable names must have a verb index, an argument index and a head index. To satisfy (1), a variable v_x is introduced whose value is an integer indicating the number of arguments the verb has. The righthand side of such a v equation sums up variables that represent verb-argument-head assignments. These variables are binary (indicator functions), they realize the variables $X_1, \ldots, X_n$ of the general ILP specification given above. Their format is $v_xa_ih_j$ with

$$\begin{array}{lllll} 1 & \leq & \text{x} & \leq & \mid \text{verbs} \mid \\ 1 & \leq & \text{i} & \leq & \mid \text{argument positions} \mid \\ 1 & \leq & \text{j} & \leq & \mid \text{heads} \mid \end{array}$$

For example, if the ILP algorithm assigns $v_2a_1h_3$ the value 1, the first argument of the second verb predicate is said to be filled by head h_3. We also have to specify variables that consume heads that are not consumed by any verb (i.e. non-arguments), and we have to determine the weights of an assignment decision (see section 9).

The full specification of the ILP formulation of the assignment problem is: [1]

[1] Please note that $v_xa_ih_j$ is one (!) variable name and not a multiplication of v_x, a_i, and h_j.

(C1a) an argument consumes at most one head

$$\sum_j v_x a_i h_j \leq 1, \quad \forall i, x$$

(C1b) a_1 of each verb consumes exactly one head

$$\sum_j v_x a_1 h_j = 1, \quad \forall x$$

(C2) a head is attached at most once (to an argument)

$$\sum_x \sum_i v_x a_i h_j \leq 1, \quad \forall j$$

(C3) the number of arguments is at least one and at most three: $1 \leq v_x \leq 3, \ \forall x$

(C4) the argument assignment of a predicate is:

$$v_x = \sum_i \sum_j v_x a_i h_j$$

(C5) all heads are consumed by predicate variables and non-argument variables (z_i)

$$v_1 + \ldots + v_{|verbs|} + z_1 + \ldots + z_{|heads|} = |heads|$$

(C6) a head is either an argument or a non-argument:

$$z_j + \sum_x v_x a_i h_j = 1, \quad \forall i, j$$

(C7) the impact of the argument assignment as specified in C4 is:

$$i_{v_x} = \sum_i \sum_j w_{v_x a_i h_j} * v_x a_i h_j$$

(C8) the impact of the non-argument assignment is:

$$i_z = \sum_j w_{z_j} * z_j$$

Given such a set of equations where all coefficients are instantiated, the objective function is:

$$max: \ i_z + \sum_x i_{v_x}$$

As a side effect of the maximization, all indicator variables are instantiated, either to 0 (*not chosen*) or to 1 (*chosen*).

cc_1 the case of the head - if there is none (e.g. verbal heads), we use the syntactic label instead

cc_2 the distance from the verb predicate v_x to a predecessor node (the anchor) which dominates the head h_i.

- if the mother of v_x is the anchor, then the distance is set to 1
- if the grandmother of v_x is the anchor (without crossing an 'S' node), the distance is set to 2
- otherwise, the distance is infinite

cc_3 a coordination flag that indicates whether h_i is part of a coordination or not.

Figure 11: Contextual Criteria

9 The Weighting Scheme

We use conditional probabilities to compute the weights of the indicator variables: $P(v_x a_j h_i | contextual\ criteria)$. The contextual criteria are given in Fig. 11.

As usual, independence is assumed:

$$w_{v_x a_j h_i} = P(v_x a_j h_i | cc_1, cc_2, cc_3) = \prod_{k=1}^{3} P(v_x a_j h_i | cc_k)$$

These probabilities are estimated with maximum likelihood (we do some smoothing as well), e.g.

$$P(v_x a_1 h_i | case_{h_i} = nom) = \frac{freq(a_1 \wedge case = nom)}{freq(case = nom)}$$

That is, the conditional probability of being argument 1 of some verb v_x and some head h_i (given case=nom) is estimated by the frequency of argument 1 being nominative divided by the frequency of heads being nominative (whether they are argument heads or non-argument heads). The weights w_{z_i} of a non-argument head z_i (cf. C8) are estimated correspondingly.

These contextual features are simple, but we found them sufficient (see the next section for the evaluation). They are simple, but they rely on structural information of parse trees. Thus, their simplicity stems from the results of a complex machinery, namely the parser.

Criterion 1 from Fig. 11 reflects the reliability on the case feature (for nominal objects, not for verbs). Criterion 2 represents the sentence context: is there a head within the same sentence as the verb and at what distance. In rare cases, heads beyond a sentence border might be arguments as well (in e.g. elliptical constructions or given parsing errors). Finally the coordination criterion: if a head is part of a coordination, it is not a good candidate for an argument position (because the whole coordination is the argument). These criteria determine the weight of a decision. The assignment of heads to argument positions of a predicate that has got the highest weight is selected.

10 ILP Compared to the Rule Learner

Fig. 12 shows the results of the ILP approach compared to the rule learner (weighted version).

	Prec	Rec	F-meas
ILP	91.98	89.83	90.89
RL+weight	98.5	78.4	87.3

Figure 12: Comparison of ILP and Rule Learner

ILP is the winner. This is perfectly explainable, although it is a bit amazing that the simple statistical model underlying the ILP optimization task works so fine (given the small amount of data it is based on). The rule learner extracts tree structure fragments as rules. Every unseen structural encoding of semantic information lessens its recall - because there is no rule to apply. The rule learner is in a sense too fine grained: Precision rides on the back of recall. The ILP approach is coarse grained, but in balanced way: precision and recall are close together.

11 Related Work

Semantic role labeling is the topic of a number of articles, for example Gildea & Jurafsky (2002). Their algorithms are based on the FrameNet corpus, a lexical resource of more than 40.000 sentences. They use the output of the Collins Parser to train their statistical model(s). Gildea & Jurafsky (2002) rely (as we do) on structural information in the form of paths, but they do not utilize functional information (e.g. GR).

(Punyakanok et al., 2004) applied integer linear programming to semantic role labeling. They do not use a parser but a chunker and the scoring (statistical) model is provided by the SNoW learning architecture. Our model is inspired by the ILP formulation of (Punyakanok et al., 2004), but there are differences in the formalization; also the features used to train the model are different.

12 Conclusion and Outlook

We have focused on the problem of predicate argument extraction from parse trees. The performance of a rule-based learner is compared to those of an ILP approach. Both approaches have good results, the rule-based one yields a higher precision, but a lower recall than the ILP approach, which has a superior f-measure value. We found ILP a good method to succinctly express linguistic constraints. The underlying statistic model works fine, but our data base is small (1001 predicate argument structures). In order to find out whether our approaches scale up, are reliable and competitive, we have to enlarge our data base.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet Project. *Proceedings of the COLING-ACL.*

Thorsten Brants, Wojciech Skut, and Hans Uszkoreit 1999. Syntactic Annotation of a German Newspaper Corpus. *Proceedings of the ATALA Treebank Workshop*.

Sabine Buchholz. 2002. *Memory-Based Grammatical Relation Finding*. PhD thesis, Tilburg University, The Netherlands.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2004. TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide *ILK pub: ILK-0402, Tilburg University, The Netherlands.*.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics 28:3, 245-288.*

Vasin Punyakanok, Dan Roth, Wen-tau Yih, and Dave Zimak. 2004. Role Labeling via Integer Linear Programming Inference. *Proceedings of the 20th International Conference on Computational Linguistics*.

Michael Schiehlen. 2004. Annotation Strategies for Probabilistic Parsing in German. *Proceedings of the 20th International Conference on Computational Linguistics*.

Helmut Schmid. 2004. Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. *Proceedings of the 20th International Conference on Computational Linguistics*.